Einführung in dplyr-Grammatik

Daten bändigen & visualisieren mit 

B. Philipp Kleer
Methodentage 2021
11. Oktober 2021

Starten wir jetzt!

Nun tauchen wir in die Welt von **dplyr** ein. Das Paket nutzt man oft, um Datenstrukturen zu erkunden oder Transformationen vorzunehmen. Dabei gibt es einen Grundstock an Vokabeln, die über **piping** miteinander verbunden werden.

Dazu installieren wir zuerst **tidyverse**:

```
install.packages("tidyverse")
library("tidyverse")

# alternativ:
# install.packages("dplyr")
# library("dplyr")
```

`

Datensatz laden

Anschließend laden wir den Datensatz `uni` ins *environment*.

```
uni <- readRDS("../datasets/uni.rds")
# oder eigener Pfad, wenn nicht in der Cloud
```

Wir verschaffen uns einen Überblick über den Datensatz:

```
head(uni,
     n = 10
     )
```

`

Datensatz im Überblick

```
   ID mot                study      city distance abi term
1   1   8   Political Science Frankfurt       NA 1.6    3
2   2   4           Sociology Frankfurt       36 3.0    5
3   3   2   Political Science   Marburg       56 2.1    4
4   4   1           Sociology    Gießen       62 3.3    5
5   5   3          Psychology   Marburg       43 1.0    8
6   6   1   Political Science   Marburg       43 1.1    2
7   7   7 Educational Science   Marburg       39 3.8    3
8   8   0           Sociology   Marburg       44 2.0    3
9   9   6          Psychology   Marburg       38 1.3    9
10 10   6          Psychology    Gießen       59 1.6    4
```

Einen Überblick über die Variablen:

```
# ID: laufende Nummer
# mot: Studienmotivation (0 <sehr niedrig> - 10 <sehr hoch>)
# study: Studienfach (1 <Political Science>, 2 <Sociology>, 3 <Educational Science>, 4 <Psychology>)
# city: Studienort (1 <Gießen>, 2 <Marburg>, 3 <Frankfurt>)
# distance: Anfahrtsdauer zur Uni in Minuten
# abi: Abiturnote
# term: Fachsemester
```

Grundvokabeln in dplyr

In `dplyr` gibt es nicht viele Vokabeln, die aber effektiv miteinander verbunden werden können, um Daten zu sortieren bzw. zu manipulieren.

Die Grundvokabeln lernen wir jetzt im Folgenden erstmal ohne *piping* kennen:

- `select()`
- `slice()`
- `filter()`
- `arrange()`
- `mutate()`
- `summarise()` / `summarize()`
- `group_by()`

`

select()

Mit **select()** wählen wir Spalten aus, die uns angezeigt werden

```
select(uni,
       c(mot,
         term
         )
       )
```

Filter:

	mot	term
1	8	3
2	4	5
3	2	4
4	1	5
5	3	8

Showing 1 to 5 of 1,000 entries

slice()

Demgegenüber können wir mit **slice()** Zeilen auswählen, also Fälle:

```
slice(uni,
      50:55
      )
```

```
  ID mot              study      city distance abi term
1 50   7 Educational Science Frankfurt       18 2.2    9
2 51   5   Political Science    Gießen       56 1.6    3
3 52   0 Educational Science   Marburg       48 1.2    6
4 53   1   Political Science   Marburg       57 3.4    3
5 54   2   Political Science   Marburg       69 2.6    3
6 55   6           Sociology Frankfurt       22 3.2    5
```

`

filter()

Mit `filter()` können wir spezifische Fälle des Datensatzes auswählen. Zur Erinnerung die logischen Verknüpfungen in R:

- logisches und: `&`
- logisches oder: `|`
- logisches gleich: `==`
- logisches ungleich: `!=`
- logisches größer: `>`
- logisches kleiner: `<`
- logisches kleiner gleich: `<=`
- logisches größer gleich: `>=`

`

filter()

Zur Anwendung:

```
filter(uni,
       city == "Gießen"
       )
```

`

	ID	mot	study	city	distance	abi	term
1	4	1	Sociology	Gießen	62	3.3	5
2	10	6	Psychology	Gießen	59	1.6	4
3	11	4	Psychology	Gießen	69	2.4	2
4	20	1	Psychology	Gießen	63	3.0	7
5	21	2	Educational Science	Gießen	74	2.5	9
6	22	6	Educational Science	Gießen	60	2.2	4
7	24	0	Sociology	Gießen	75	3.3	6
8	25	2	Sociology	Gießen	56	3.8	4
9	28	2	Sociology	Gießen	82	2.7	9

arrange()

Mit **arrange()** können wir den Datensatz sortieren.

```
arrange(uni,
        abi
        )
```

`

	ID	mot	study	city	distance	abi	term
1	5	3	Psychology	Marburg	43	1.0	8
2	14	7	Political Science	Marburg	44	1.0	8
3	87	9	Educational Science	Gießen	54	1.0	4
4	106	9	Political Science	Frankfurt	35	1.0	7
5	238	1	Political Science	Marburg	51	1.0	6
6	271	1	Psychology	Gießen	80	1.0	8
7	274	6	Political Science	Marburg	59	1.0	9
8	300	6	Political Science	Marburg	48	1.0	3
9	348	1	Educational Science	Frankfurt	NA	1.0	6

arrange()

Die Sortierung ist dabei immer aufsteigend. Dies kann man über die Funktion `desc()` ändert (**desc**ending):

```
arrange(uni,
        desc(abi)
        )
```

`

	ID	mot	study	city	distance	abi	term
1	61	6	Psychology	Marburg	42	4.0	4
2	99	3	Sociology	Frankfurt	25	4.0	9
3	126	6	Sociology	Marburg	42	4.0	5
4	148	5	Educational Science	Gießen	64	4.0	9
5	154	8	Psychology	Marburg	47	4.0	4
6	156	7	Psychology	Frankfurt	21	4.0	9
7	181	8	Sociology	Frankfurt	36	4.0	8
8	189	1	Psychology	Frankfurt	NA	4.0	2
9	232	2	Sociology	Gießen	72	4.0	9

arrange()

Alternativ kann man auch einfach ein Minuszeichen vor die Variable, nach der sortiert werden soll, setzen:

```
arrange(uni,
        -abi
        )
```

`

	ID	mot	study	city	distance	abi	term
1	61	6	Psychology	Marburg	42	4.0	4
2	99	3	Sociology	Frankfurt	25	4.0	9
3	126	6	Sociology	Marburg	42	4.0	5
4	148	5	Educational Science	Gießen	64	4.0	9
5	154	8	Psychology	Marburg	47	4.0	4
6	156	7	Psychology	Frankfurt	21	4.0	9
7	181	8	Sociology	Frankfurt	36	4.0	8
8	189	1	Psychology	Frankfurt	NA	4.0	2
9	232	2	Sociology	Gießen	72	4.0	9

mutate()

Mit **mutate()** werden neue Variablen geschaffen.

Zum Beispiel könnten wir eine Variable schaffen, die den Abstand zum Mittelwert in der Variable `abi` misst.

Wichtig: Wir haben zwar hier die Variable `abiDist` gespeichert, aber diese nicht im Datensatz gespeichert.

```
mutate(uni,
       abiDist = abi - mean(abi,
                            na.rm = TRUE
                           )
       )
```

	ID	abi	abiDist	mot	study	city	distance	ter
1	1	1.6	-0.9347	8	Political Science	Frankfurt	NA	
2	2	3.0	0.4653	4	Sociology	Frankfurt	36	
3	3	2.1	-0.4347	2	Political Science	Marburg	56	
4	4	3.3	0.7653	1	Sociology	Gießen	62	
5	5	1.0	-1.5347	3	Psychology	Marburg	43	
6	6	1.1	-1.4347	1	Political Science	Marburg	43	
7	7	3.8	1.2653	7	Educational Science	Marburg	39	
8	8	2.0	-0.5347	0	Sociology	Marburg	44	
9	9	1.3	-1.2347	6	Psychology	Marburg	38	

`

mutate() & case_when()

Bei der Erstellung kategorieller Variablen muss man zusätzlich die Funktion **case_when()** nutzen. **case_when()** funktioniert wie eine Aneinanderreihung von *if*-Bedingung, wobei die spezifischste Bestimmung zuerst kommen sollte. (spezifisch -> allgemein).

Im Beispiel schaffen wir eine Dummy-Variable, die anzeigt, ob die Person in Marburg studiert (`1`) oder nicht (`0`).

Die Grammatik in **case_when()** ist wie folgt:

```
case_when(Fallauswahl ~ neuer Codewert)
```

Im Beispiel:

```
mutate(uni,
       dumPum = case_when(city == "Marburg" ~ 1, # "Fallauswahl" ~ "neuer Codewert"
                          city == "Gießen" ~ 0,
                          city == "Frankfurt" ~ 0
                          )
       )
```

`

mutate() & case_when()

```
  ID mot               study      city distance abi term dumPum
1  1   8   Political Science Frankfurt       NA 1.6    3      0
2  2   4           Sociology Frankfurt       36 3.0    5      0
3  3   2   Political Science   Marburg       56 2.1    4      1
4  4   1           Sociology    Gießen       62 3.3    5      0
5  5   3          Psychology   Marburg       43 1.0    8      1
6  6   1   Political Science   Marburg       43 1.1    2      1
7  7   7 Educational Science   Marburg       39 3.8    3      1
8  8   0           Sociology   Marburg       44 2.0    3      1
9  9   6          Psychology   Marburg       38 1.3    9      1
```

`

mutate() & case_when()

Auch hier könnten mehrere Bedingungen verknüpft werden: So möchten wir einen Dummy schaffen, der anzeigt, ob eine Person in Marburg Erziehungswissenschaften studiert.

Wir würden wie folgt beginnen:

```
mutate(uni,
       dumPumEs = case_when(city == "Marburg" & study == "Educational Science" ~ 1
                            )
       )
```

Wenn man nicht alle verschiedenen Kombinationen eingeben möchte und zum Beispiel nur eine von Interesse ist, kann man mit `TRUE ~ 0` allen restlichen Fällen direkt einen Wert zuordnen (aber nur denselben Wert!). Alle Kombinationen, die nicht vor `TRUE ~ 0` definiert wurden, erhalten automatisch den in der `TRUE` -Zeile definierten Wert.

```
mutate(uni,
       dumPumEs = case_when(city == "Marburg" & study == "Educational Science" ~ 1,
                            TRUE ~ 0
                            )
       )
```

`

mutate() & case_when()

```
  ID mot              study      city distance abi term dumPumEs
1  1   8  Political Science Frankfurt       NA 1.6    3        0
2  2   4          Sociology Frankfurt       36 3.0    5        0
3  3   2  Political Science   Marburg       56 2.1    4        0
4  4   1          Sociology    Gießen       62 3.3    5        0
5  5   3         Psychology   Marburg       43 1.0    8        0
6  6   1  Political Science   Marburg       43 1.1    2        0
7  7   7 Educational Science   Marburg       39 3.8    3        1
8  8   0          Sociology   Marburg       44 2.0    3        0
9  9   6         Psychology   Marburg       38 1.3    9        0
```

`

summarise()

Mit **summarise()** (oder **summarize()**) können vereinfacht erste Einblicke in die Daten erfolgen. So könnten wir uns z.B. den Mittelwert von `term` ausgeben lassen.

```
summarise(uni,
          mean(term)
          )
```

```
  mean(term)
1      6.069
```

In **summarise()** können verschiedene Funktionen genutzt werden, die auf die Variablen im Datensatz angewendet werden können. Auch können direkt mehrere Werte ausgegeben werden.

Wichtig: Das Ausgabe-Format ist immer ein *tibble*.

```
summarise(uni,
          mean(term),
          mean(mot)
          )
```

```
  mean(term) mean(mot)
1      6.069     4.561
```

summarise_if()

Die Unterfunktion **summarise_if()** bietet dazu die Möglichkeit leicht auf eine Gruppe von Variablen Funktionen anzuwenden, also zum Beispiel auf alle numerischen Variablen:

```
summarise_if(uni,
             is.numeric,
             list(mean = mean,
                  sd = sd
                  )
             )
```

```
  ID_mean mot_mean distance_mean abi_mean term_mean    ID_sd   mot
1   500.5    4.561            NA   2.5347     6.069 288.8194 2.897
                          abi_sd  term_sd
                       1 0.8741453 2.578544
```

Wer weiß, warum hier teils `NA` angezeigt wird?

`

summarise_at()

Die Unterfunktion **summarise_at()** bietet die Möglichkeit nur bei bestimmten Variablen die Funktion anzuwenden:

```
summarise_at(uni,
             vars(mot,
                  abi,
                  term
                  ),
             list(mean = mean,
                  sd = sd
                  )
             )
```

```
  mot_mean abi_mean term_mean   mot_sd    abi_sd  term_sd
1    4.561   2.5347     6.069 2.897011 0.8741453 2.578544
```

`

group_by()

Mit **group_by()** kann der Datensatz gruppiert werden, also zum Beispiel nach eine kategoriellen Variable. In `uni`-Datensatz zum Beispiel nach `study`:

```
group_by(uni,
         `study`
         )
```

Was sehen wir?

`

```
# A tibble: 1,000 × 7
# Groups:   study [4]
     ID   mot study             city      distance   abi  term
  <int> <int> <fct>             <fct>        <dbl> <dbl> <int>
1     1     8 Political Science Frankfurt       NA   1.6     3
2     2     4 Sociology         Frankfurt       36   3       5
3     3     2 Political Science Marburg         56   2.1     4
4     4     1 Sociology         Gießen          62   3.3     5
5     5     3 Psychology        Marburg         43   1       8
6     6     1 Political Science Marburg         43   1.1     2
```

group_by()

group_by() macht nichts weiter als die Daten zu gruppieren, die Ausgabe verändert sich dabei erstmal nicht. Erst in Kombination mit weiteren Funktionen, wird dies sichtbar:

```
summarize(group_by(uni,
                   study
                   ),
          mean(term)
          )
```

```
# A tibble: 4 × 2
  study                `mean(term)`
  <fct>                       <dbl>
1 Political Science            5.93
2 Sociology                    6.13
3 Educational Science          6.22
4 Psychology                   5.96
```

Jetzt haben wir für jeden Studienort einen Mittelwert für das Fachsemester (`term`).

Wichtig: Wenn Daten gespeichert oder übergeben werden, sollte am Ende die Befehlskette immer mit `ungroup()` enden, um die Datenteilung nicht zu übergeben!

`

Piping `%>%`

Pipes mit tidyverse

Mit den sogenannte *pipes* können Ergebnisse von Ausführungsschritten weitergegeben werden. Dies ist vorteilhaft, da so verschiedene Schritte direkt ausgeführt werden können. Auch kann so Code oftmas leichter nachvollzogen werden.

Den *pipe*-Operator in `tidyverse` ist `%>%` und kann einfach per Tastenkürzel hinzugefügt werden (`Strg` / `Cmd` + `Shift` + `M`).

Seit R Version 4.0 gibt es den Pipe-Operator auch in RBase (`\>`), daher diese beiden nicht verwechseln.

`

Pipes

Hier mal ein Beispiel: Das Ziel ist es eine Variable zu erstellen, die den Abiturschnitt pro Uni-Stadt ausgibt. Das könnte die Frage beantworten, ob besonders gute Schüler:innen einen der drei Studienorte präferieren.

`

Beispiel Pipes

Die Schritte, die wir hierbei machen, sind folgende:

1. Wir geben den Datensatz `uni` weiter.
2. Wir gruppieren den Datensatz nach `city`.
3. Wir berechnen eine neue Variable `abiMean`.
4. Wir heben die Gruppierung wieder auf.
5. (bzw. 0.) Wir überspeichern den alten Datensatz.

```
uni <- uni %>%
  group_by(city) %>%
  mutate(abiMean = mean(abi,
                        na.rm = TRUE
                        )
         ) %>%
  ungroup()

table(uni$city,
      uni$abiMean)
```

```
                2.51 2.51597633136095 2.58269230769231
   Gießen          0               338                 0
   Marburg       350                 0                 0
   Frankfurt       0                 0               312
```

`

Pipes

Alternativ könnten wir uns dies auch erstmal nur ausgeben lassen.

```
uni %>%
  group_by(city) %>%
  summarize(mean = mean(abi,
                        na.rm = TRUE
                        )
            )
```

```
# A tibble: 3 × 2
  city       mean
  <fct>     <dbl>
1 Gießen     2.52
2 Marburg    2.51
3 Frankfurt  2.58
```

`

Pipes

Ein weiteres Beispiel: Wir möchten Studierende nach der Anzahl des Fachsemesters kategorisieren. Die neue Variable `termg` soll zwischen:

- Anfänger:innen (<=2 Semester)
- Erfahrene (>2 & <= 6 Semester)
- Langzeitstudierende (>6 Semester)

unterscheiden.

```
uni <- uni %>%
  mutate(termg = case_when(term <= 2 ~ "Anfänger:in",
                           term > 2 & term <= 6 ~ "Erfahrene",
                           term > 6 ~ "Langzeit"
                           )
         )
table(uni$termg)
str(uni$termg)
```

```
Anfänger:in   Erfahrene    Langzeit
         93         468         439
```

```
 chr [1:1000] "Erfahrene" "Erfahrene" "Erfah
```

`

Pipes

Etwas komplexer wäre folgende Aufgabe: Wir möchten nicht die Abweichung zum Mittelwert des Abiturs in unserer gesamten Erhebung berechnen, sondern die Abweichung zum Mittelwert der einzelnen Universitäten. Damit wir die Gruppen-Mittelwerte angezeigt bekommen, berechnen wir auch eine Variable für den Gruppen-Mittelwert.

```
uni <- uni %>%
  group_by(city) %>%
  mutate(abigm = mean(abi)) %>%
  mutate(abid = abi - abigm) %>%
  ungroup()

uni[, c("ID",
        "abi",
        "city",
        "abigm",
        "abid"
        )
    ]
```

```
# A tibble: 1,000 × 5
      ID   abi city      abigm   abid
   <int> <dbl> <fct>     <dbl>  <dbl>
 1     1   1.6 Frankfurt  2.58 -0.983
 2     2   3   Frankfurt  2.58  0.417
 3     3   2.1 Marburg    2.51 -0.41
 4     4   3.3 Gießen     2.52  0.784
 5     5   1   Marburg    2.51 -1.51
 6     6   1.1 Marburg    2.51 -1.41
 7     7   3.8 Marburg    2.51  1.29
 8     8   2   Marburg    2.51 -0.51
 9     9   1.3 Marburg    2.51 -1.21
10    10   1.6 Gießen     2.52 -0.916
# … with 990 more rows
```

`

Pipes

Alternativ könnten wir die Daten auch hierarchisch nach Standort und Studienfach gruppieren und uns dann einfach die unterschiedlichen Mittelwerte mit **summarise()** ausgeben lassen:

--

```
mCityStudy <- uni %>%
  group_by(city,
           study
           ) %>%
  summarise(mean(abi))

mCityStudy
```

`

```
# A tibble: 12 × 3
# Groups:   city [3]
   city      study                `mean(abi)`
   <fct>     <fct>                      <dbl>
 1 Gießen    Political Science           2.38
 2 Gießen    Sociology                   2.51
 3 Gießen    Educational Science         2.57
 4 Gießen    Psychology                  2.57
 5 Marburg   Political Science           2.48
 6 Marburg   Sociology                   2.51
 7 Marburg   Educational Science         2.49
 8 Marburg   Psychology                  2.56
 9 Frankfurt Political Science           2.34
10 Frankfurt Sociology                   2.68
11 Frankfurt Educational Science         2.60
12 Frankfurt Psychology                  2.71
```

Versuchen wir es zusammen zu lösen!

Versucht euch mit dem Grundvokabular an folgenden Aufgaben in den Breakout-Rooms oder allein:

1. Teile den Datensatz uni in drei Datensätze, die jeweils nur eine Universitätsstadt inkludieren.

2. Berichte die durchschnittliche Semesterzahl pro Uni und Studiengang!

3. Berechne eine Variable, die die Abweichung von der durchschnittlichen Semesterzahl nach Studienfach angibt.

`

Lab Task

In der nächsten halbe Stunde sollt ihr euch in Gruppen (Breakout-Rooms) oder einzeln an den folgenden Aufgaben versuchen. Es müssen nicht alle Aufgaben in der Zeit geschafft werden, es geht viel mehr um die Auseinandersetzung mit dem neuen Vokabular.

Nutzt dazu den Datensatz `pss` (Panem Social Survey).

1. Filter den Datensatz, so dass ein Subset nur mit Personen aus Distrikt 1 entsteht. Lass dir mit **pipes** jeden 150. Fall anzeigen.

2. Filter den Datensatz, so dass ein Subset entsteht, dass keine Personen aus Distrikt 1 oder Distrikt 5 beinhaltet. Lass dir mit **pipes** die letzten 50 Fälle anzeigen.

3. Filter den Datensatz, so dass ein Subset mit Personen entsteht, die entweder in Distrikt 7 oder nicht in Distrikt 12 leben. Lass dir im Pipen die ersten 15a Fälle anzeigen.

4. Woher kommen die 10 ältesten Personen.

5. Wie stufen sich die 10 jüngsten personen auf der Links-Rechts-Selbsteinschätzung ein.

6. Gruppiere den Datensatz nach Distrikten und lasse dir deskriptive Werte für die Links-Rechts-Selbsteinschätzung ausgeben.

Das wars!